Task 3 - Card Game

Analysis

# Success Criteria

1. There must be 30 cards each card must have:

* A colour assigned to it, this colour must be RED, BLACK or YELLOW
* For each colour there must be ten numbers, one for each card.
* There should be 10 RED cards, 10 BLACK cards and 10 YELLOW cards.

1. These cards must be shuffled and stored in the deck (a list called “deck”)
2. Every turn each player takes a card from the top of the deck. Player 1 starts by taking a card and then Player 2 follows suit.
3. The two cards picked up by the 2 players are compared by:

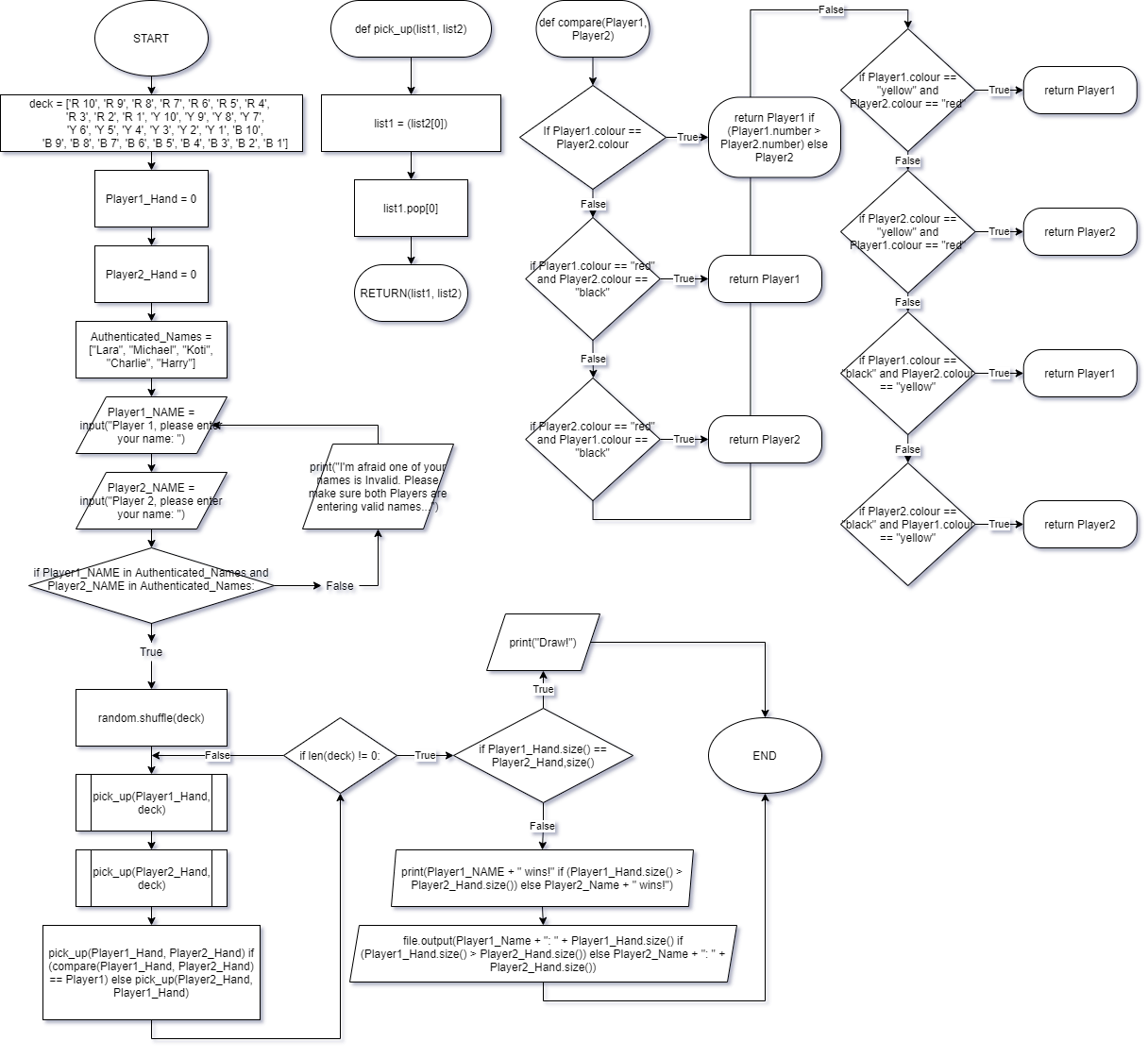
* Comparing the colours of the two cards, if they are the same then:
  + Compare the size of the number, the player with the higher number wins both cards
* If they are different colours then they are compared with the table shown below:

|  |  |  |
| --- | --- | --- |
| **Card** | **Card** | **Winner** |
| **Red** | **Black** | **Red** |
| **Yellow** | **Red** | **Yellow** |
| **Black** | **Yellow** | **Black** |

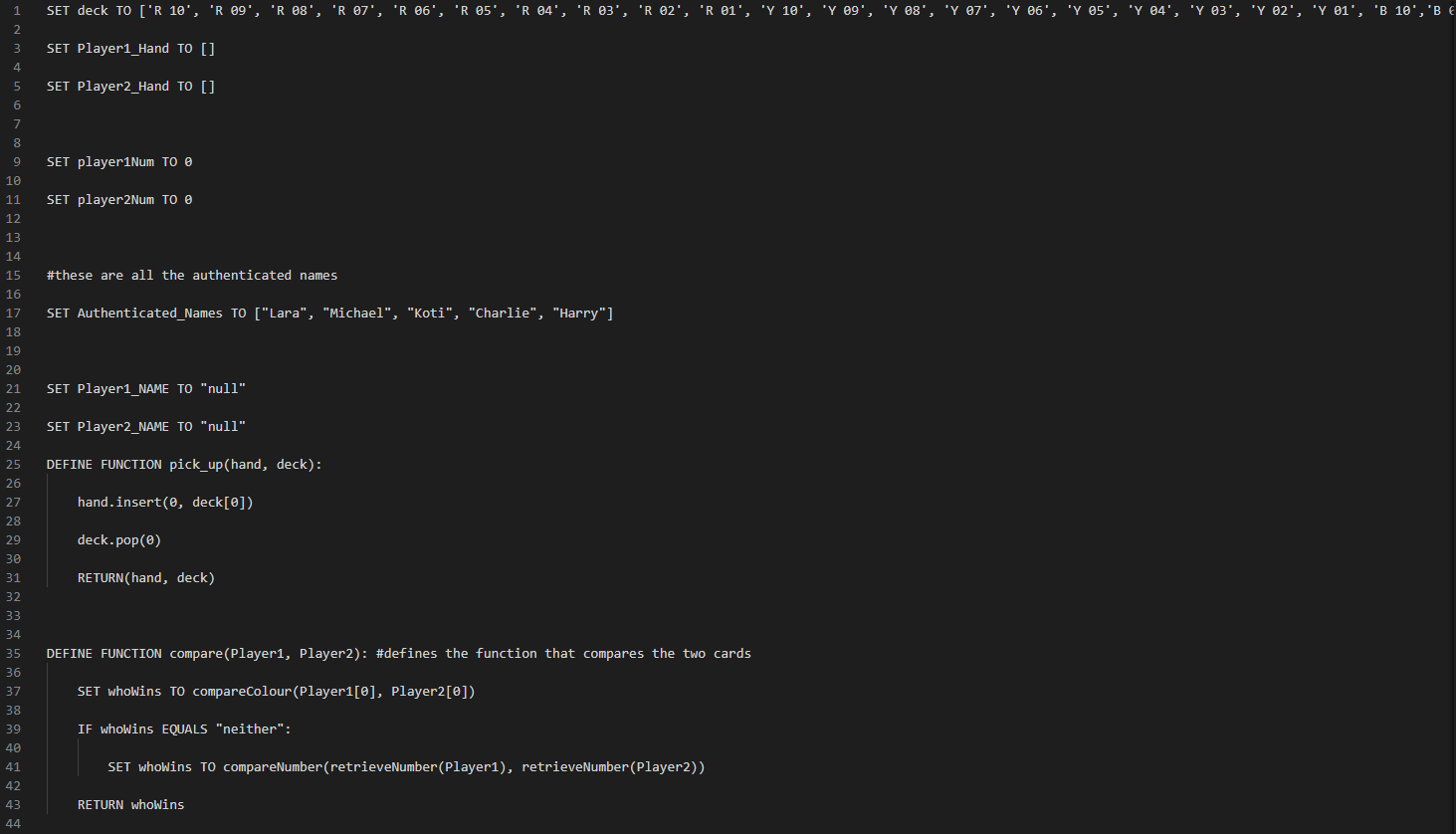
1. The winner of each round keeps both cards played
2. At the start, let the two players enter their details, which are then checked by the program to ensure both players are authorised players.
3. Once the game is finished the amount of cards the winner has is outputted and saved to an external save file with the authenticated players name.
4. The program should have the ability to display the name and quantity of cards of the 5 players with the highest quantity of cards, from the external file.

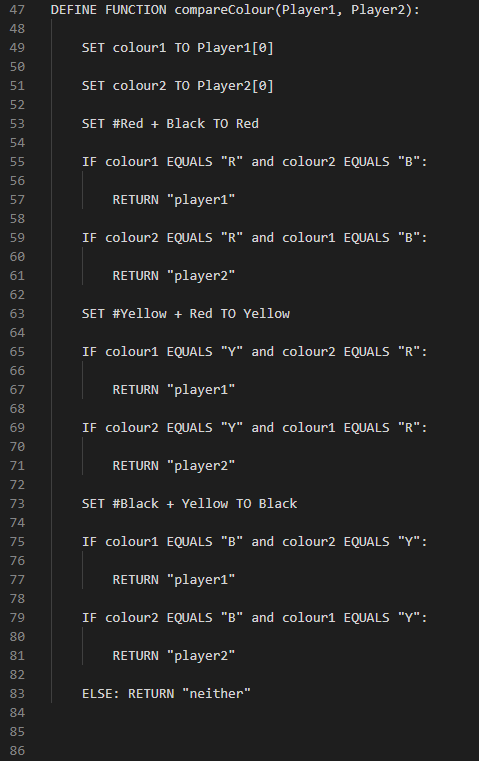
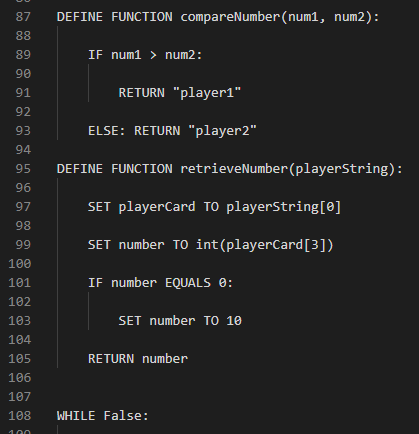
# Justification of Design and initial Code Ideas:

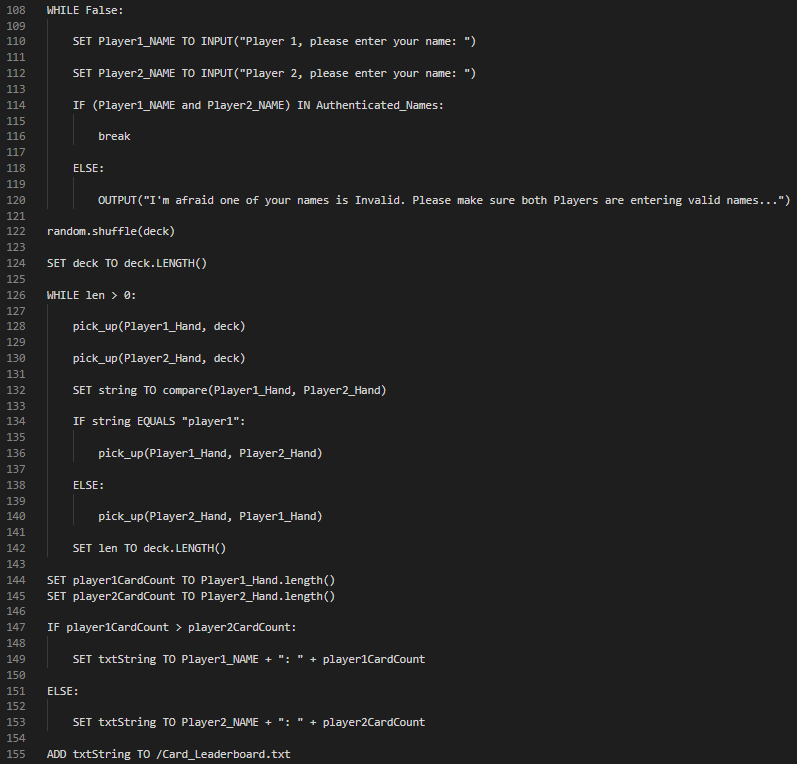
1. I will create a list called ‘deck’ which will contain 30 ‘Cards’ (strings) each assigned with a colour. 10 Items (1-10) will be assigned the Red colour, 10 Items (1-10) will be assigned the Yellow colour and 10 Items (1-10) will be assigned the Black colour.
2. Player 1 and Player 2 will be asked for their information, this information will be compared to a variable which will contain the needed information to access the game. It will also be accessed at the end of the game to be saved into the external file if either player wins.
3. I will use the shuffle function from the Random Library to randomize the items in the list ‘deck’, I will then use the indexs of each item to retrieve the ‘card’ from the top of the deck.
4. Using the index of each item, the program will retrieve a card from the top of the ‘deck’, this card will then be subtracted from the list ‘deck’ and added to ‘Player#\_Picked’ (hash being the number of the Player) Then this will happen again except the Card will be added to the other players ‘Picked’ variable.
5. Then the two cards in the players 'Picked’ variable will be compared.
6. This will then repeat for the other player until the entire ‘deck’ has been depleted.



# PSEUDO Code





# Python Code

**Day 2**

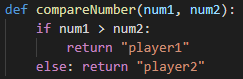


This function compares two colours by taking two strings (and ) reading the first value from each string ( and ) and comparing them to the table.



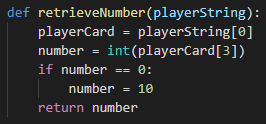
Then, the program returns a string containing the player that won (this function assumes that colour1 comes from player1). Something that I changed later to make more obvious:

This takes two numbers, (num1 being player1 and num2 being player2) and returns a string containing the player with the highest number.



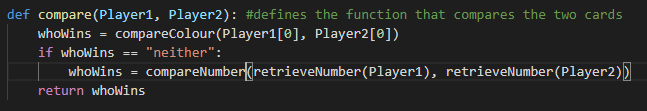
This takes the players string (so the card) and retrieves the number contained in it.

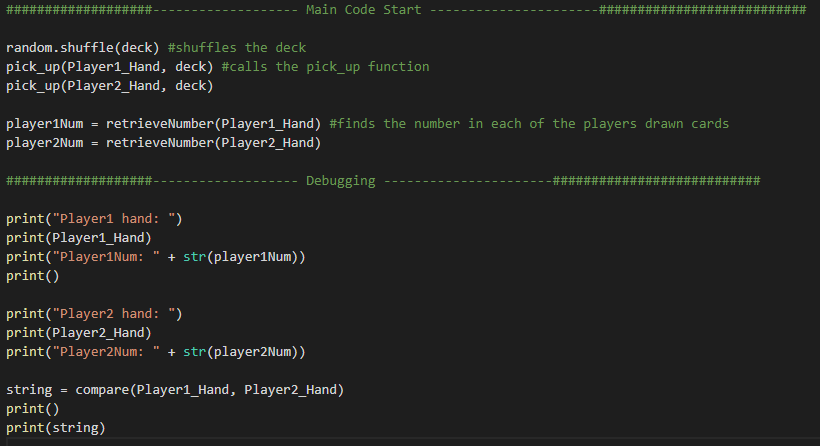
It also converts the character at: playerCard, index 3 to an integer so it can check if the number == 0. This means the program can account for the number 10 without using a complicated algorithm. In hindsight, I could’ve made the range of number 0-9, which could also make the program more efficient. (May add this later)



This is where all of the comparing functions are called from, this allows for me to easily determine where problems in the code are and allows me to repeatedly call the function with little effort.

The function takes the 1st item in each of the players ‘deck’s and compares the colour (explained above) then if neither player wins from their colour alone, then each players number is compared. The variable whoWins is temporary (within the function) and allowed for debugging when determining problems with the function later.





This is the main area of the program, everything above is where I declare functions and variables.

The code starts by shuffling the array ‘deck’ and calling the ‘pick\_up’ function (explained below). Then the number on each of the players cards is retrieved (using the retrieveNumber function explained above).

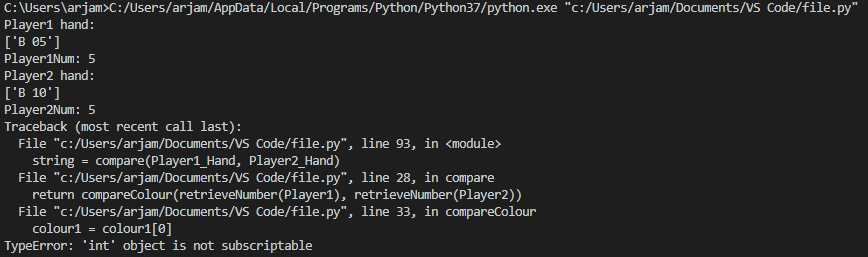
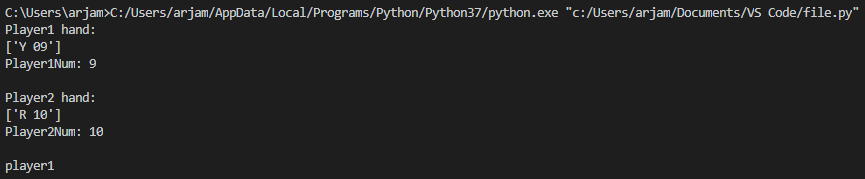
Then some debugging takes place. This allowed me to check for any logic/syntax errors, of which one was found (below).

Finally, a variable ‘string’ (temporary name) is declared as the result of the function compare() (also highlighted above)

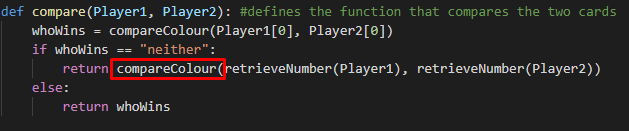
This is the output I got from a ‘previous’ version of the code (before I fixed the problem). As you can see, the program runs fine when comparing the colours of the two players’ cards.

However when the two colours are the same and the numbers should be compared, an error occurs.

**The error reads:** ‘int’ object is not subscriptable as a TypeError.



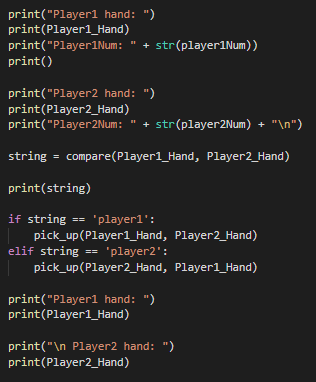
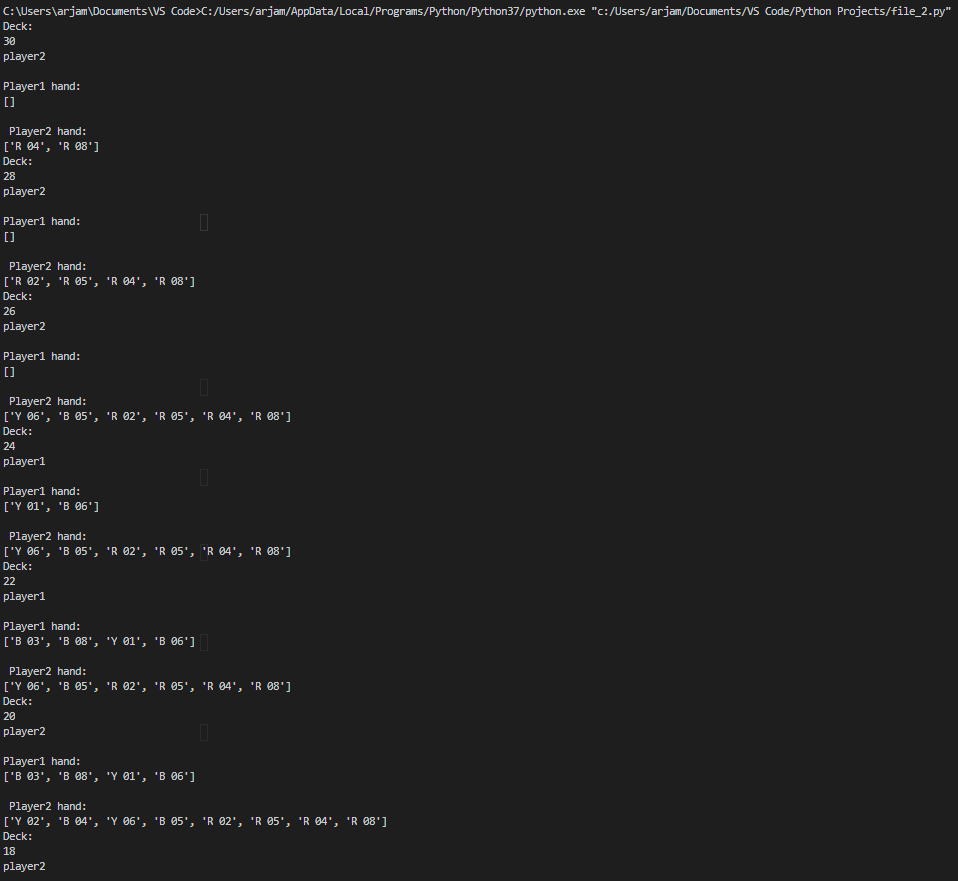
Turns out, I had accidentally called the wrong function when meaning to compare the numbers. I proceeded to fix this and no more errors occurred (a good sign)



**Day 2 of Python Coding Summary**

The code now runs absolutely fine and it’s clear that the program will complete the task with relative ease however I still need to make the ‘main’ part of the program repeats itself until all the cards in the deck have been depleted and improve the efficiency by changing the range of numbers from 1-10 to 0-9.

**Day 3**



This was the output once the entire program had run through the while loop until the ‘deck’ was exhausted.

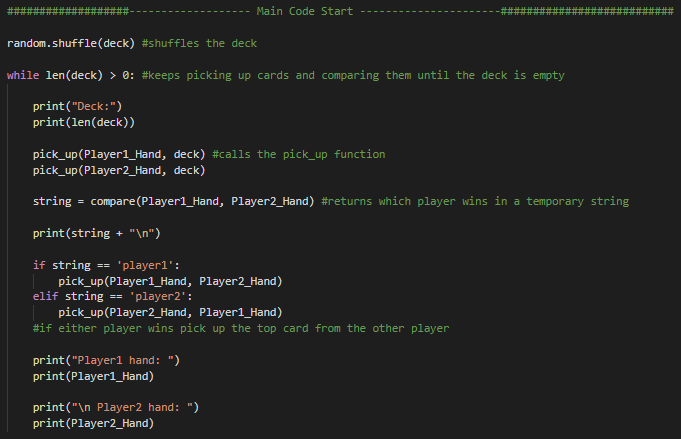
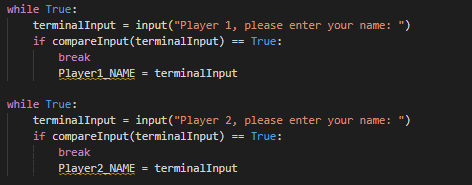
The repeating part I added was a while loop that looped continuously until the length of the ‘deck’ was 0. Then it would stop.

The current code prints player 1’s hand, then player1’s number on the top card in their hand and does the same for player 2. This was only added for debugging

Then the two hands are compared ONCE and the outcome is outputted. I then used the ‘pick\_up’ function I coded earlier to allow the winner to take the other players top card.

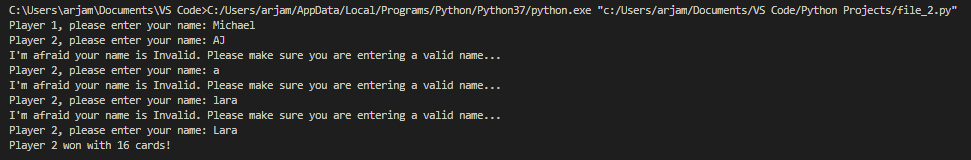
Then, for debugging, I outputted each player’s hand.

I ran this code and ran into no errors, this means the main function of the program was complete, now all I needed to do was repeat the process until the ‘deck’ was exhausted.





I then proceeded to change my initial player name input by using a repeated function. This meant that the user would have a better idea about which player had the invalid name. Since before, an error would only be returned if either name was invalid.

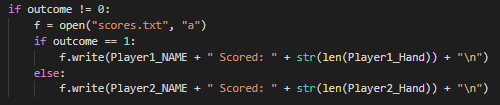
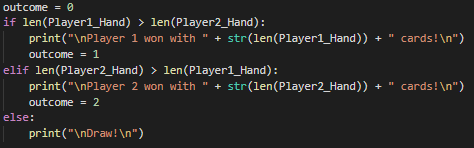


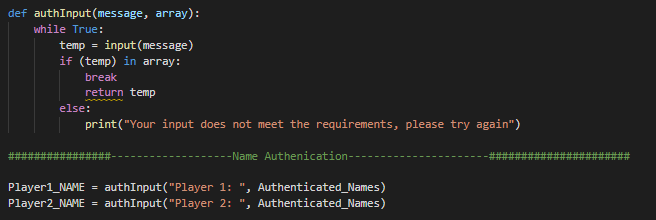
This code:

Outputs the winner and their card count.

Then using the variable ‘outcome’, I can store the winner’s player number in memory.

Then, if the outcome isn’t 0 (which is the draw condition) then the players name and score is written to a separate text document called ‘scores’

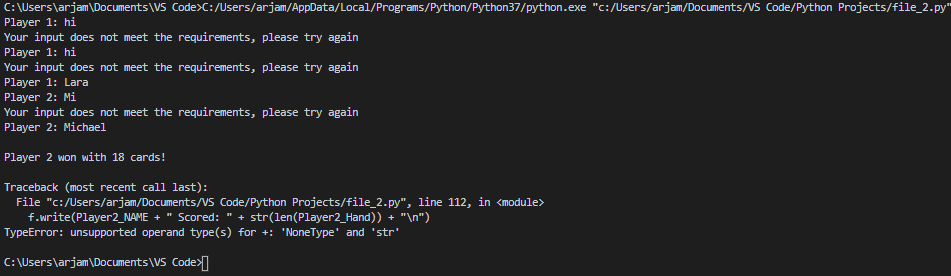


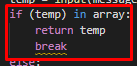


I decided to turn this into a re-useable function so I could use it later in the program, this function displays a message, takes an input and checks if the inputted string is in the array given when the function is called.

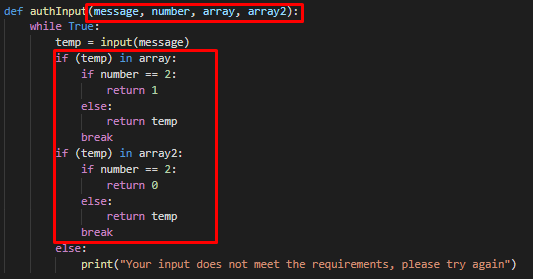
In the player name input, I used the previous array of ‘Authenticated\_Names’.

However, when I ran this new code, I got a type error. I checked my code and realized that I hadn’t returned the input until after I broke the while loop. This meant that nothing was being returned by the function and the Players Name was just ‘null’ without a type.





I then needed to ask the user if they wanted to see the top 5 scorers. I wanted to use this authInput function to allow for multiple versions of yes and no, to allow for user error and preference. However, the function could only check if the input was in one array. So I decided to allow the function to take at least 2 possible arrays to be compared:



Then I changed the original auth input to use a nullArray and specify that they only wanted to compare with one array:

This was my ‘Yes’ or ‘No’ input: 

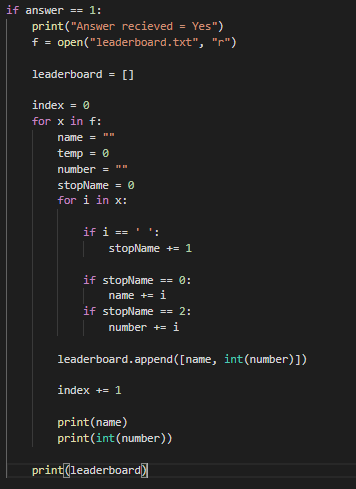
**Day 3 of Python Coding Summary**

I realise now that I should probably have separated the processes that:

* checks if the users input was correct
* asks if the input was yes or no

This part of the program is too long and complicated for what it’s worth, and could be slightly easier to read and debug if I just created another function. I will probably fix this later on. However, the other areas work brilliantly and can be super-efficient!

**Day 4**

Next, I took action on the result of the ‘Yes or No’ input from earlier.

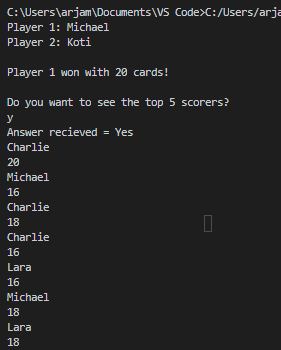
First I outputted that the computer had received the answer (mainly for debugging my previous messy functions and to accommodate for user error)

Then I opened the text document ‘leaderboard’ (same as scores.txt, just renamed) as read only.

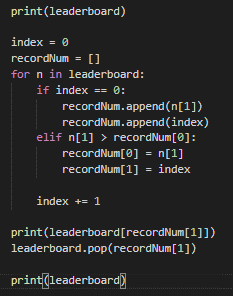
Then, I iterated through each line retrieving the name of the player on that line and their score. These values were then stored in an array called leaderboard.

For debugging, I outputted each name and number on the lines in the .txt document and finally outputted the dedicated array.

This was the output when run:



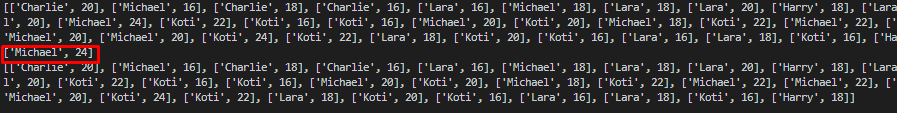
Leaderboard Array:

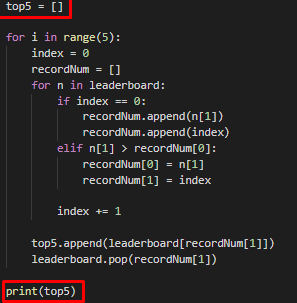


Everything checked out so I moved onwards.

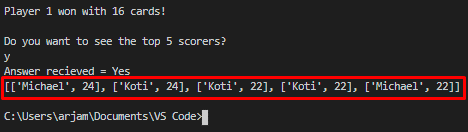
The program then looks at the first item in the leaderboard array and adds it to the array ‘recordNum’ the number contained in this array is then compared with every other number in the leaderboard array. If a larger number in the leaderboard array is found, then the name and number in recordNum is replaced. Once this process has finished, the highest score in the leaderboard array is found, stored in recordNum and removed from the leaderboard array.

This is the output of this code:



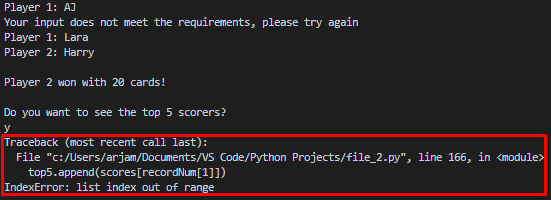
This can then be repeated 5 times so that the array ‘top5’ contains the top 5 scorers in the leaderboard array

This is the output:



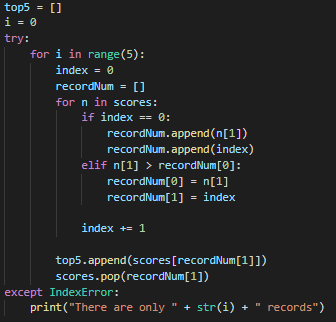
This code had a fatal flaw however. What happens when the leaderboard has less than 5 scores in total? Like this:



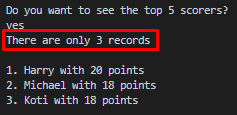
I decided to see what would happen when I tried this.

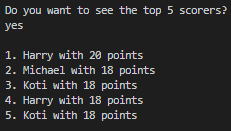
This is what the program returned:

**Day 5**

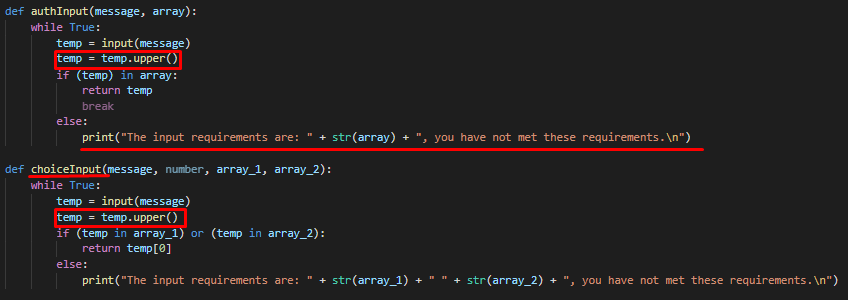
In order to combat this, I made some changes to the code:

I added an error exception, this meant that the program would attempt to retrieve 5 of the top numbers in the score.txt file. But if at any point an IndexError occurred (when the file read didn’t contain enough data to output the top 5 scorers) the program would stop, output to the user the number of records it could find and subsequently output those records.

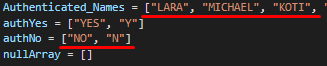


Just to test, I also made sure that this ‘error’ message didn’t show when there was enough data. Sure enough, this was no issue.

**IMPROVED INPUT FUNCTIONS**



I changed my authInput function. I realised that the input was case-sensitive and that users might find this difficult to work around in order to use my program. I also didn’t like how I merged two very different functions into one.

I split the function in two, and in both I decided to convert any user input into uppercase (temp.upper() does this) then they both do roughly the same thing (they check if the user input is in the given array/s and returns a value based off that. If this is not the case then print an error message and repeat)

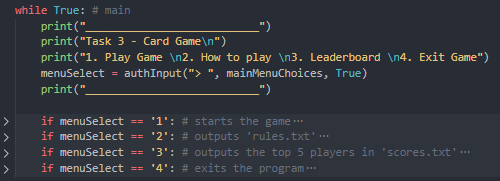
I do slightly tweak this later to add some security to the authInput functions and I realized that I could probably have taken advantage of Pythons support of function overloading and used the same function name twice.

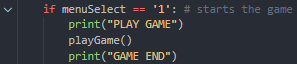
**Day 6**

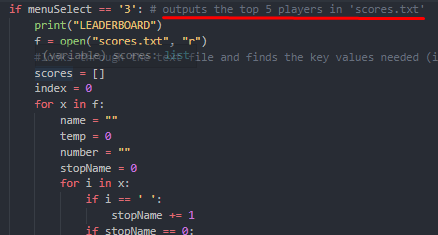
**MENU SYSTEM**

I wanted to implement a menu system to the program to improve the user interface and add some level of polish to the program. The program uses a while loop so the only way to exit the entire program is to select “Exit Game” now.

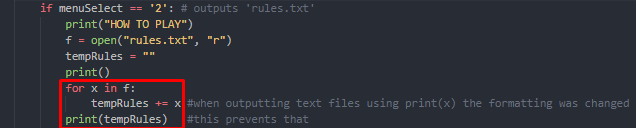
F:\Python Programming Project\Images\Screenshot_48.png

Visual Studio Code has this useful feature which allows for the collapsing of some regions of code, so if the screenshot looks weird, it’s because the code is collapsed under these if statements. The comments explain the code contained as well.

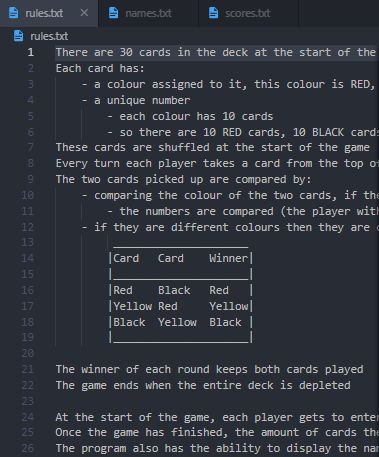
I moved the game into the function playGame() for convenience and added markers to show the user when the game had started and ended. This menu system also adds re-playability.

I also moved the score output function into the menu under “leaderboard”. Now, instead of only outputting the top 5 players at the end of the program, the user can ask for the top 5 scorers whenever they want!

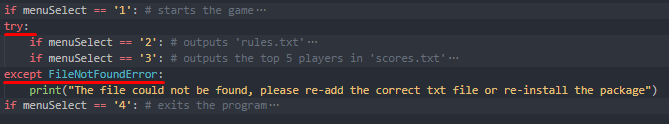
I also added a ‘rule book’ which outputs the rules of the game when selected in the menu.

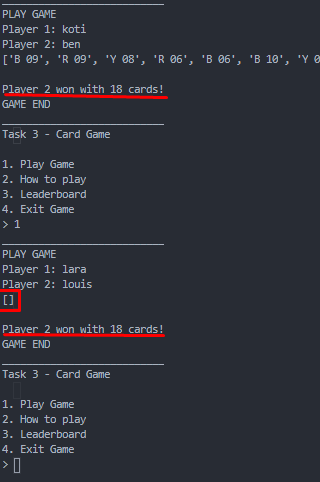


The code outlined was my way of ensuring python would output the text file exactly how it is written. The normal way python outputted it added weird indentation in some cases.

This is part of the rule book:

The program also spits out some errors when it cannot find the files needed, so I added some error detection

Which outputs a more civilised message and doesn’t end the program.

**MENU SYSTEM PROBLEMS**

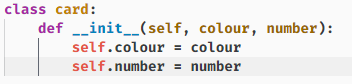
The re-playability of the main menu introduced a Logic Error which meant that the final winner was repeated over and over again

I realized this was because I forgot to reset the deck every time you start a game.

**OBJECT ORIENTED PROGRAMMING**

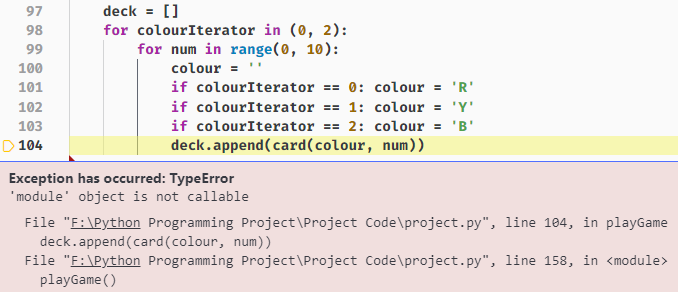
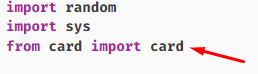
This next section outlines how I went about converting some elements of the project into an Object Oriented Program (a “technique” which I learnt which will make the whole project run more effectively)

I started off by creating a simple class in a separate python file here:

This is a simple class at the moment and will allow me to convert the deck from an array of strings to an array of card objects.

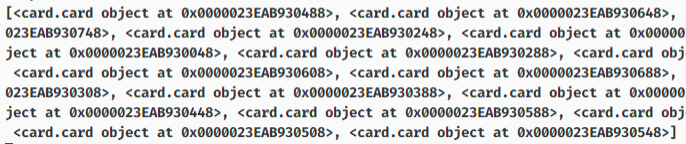
I plan to convert all the current arrays and variables into classes to make managing code easier.

I toyed with the idea of turning the entire deck into a class, but I thought that keeping the current python array would work just fine.

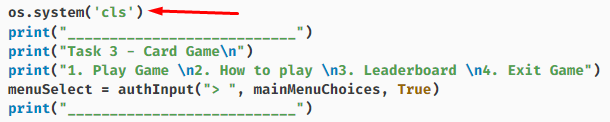
This error occurred when I tried initializing the array of card objects.

My problem was that I created my class in a different python file and forgot to import the class into the main python file.

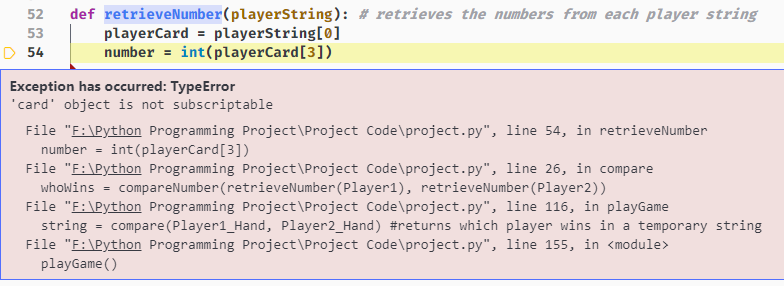
Now when I print the deck, it outputs this:



Which is what I expect! It shows that the deck is filled with card objects at different points in memory, which shows that the class system works. Now I need to make sure the rest of the program handles this change.



I’ve used os.system(“cls”) to clear the python console at certain points in my menu system to improve overall clarity

Another error, this was because I hadn’t changed my functions to work with this new format and the retrieveNumber function was getting confused

All I need to do is fix this:



By turning it into this:



Which means I can also get rid of the retrieveNumber function and replace the compareNumber function with python’s ternary operator



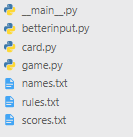
I also moved the authenticated names to a text file and added error handling to ensure the game wouldn’t run without the text file.



Now this all works perfectly and I have improved my code readability and efficiency!

**File Organisation**

Next I decided to learn how to separate the program into separate python files for better organization:

 card.py – contains the card class

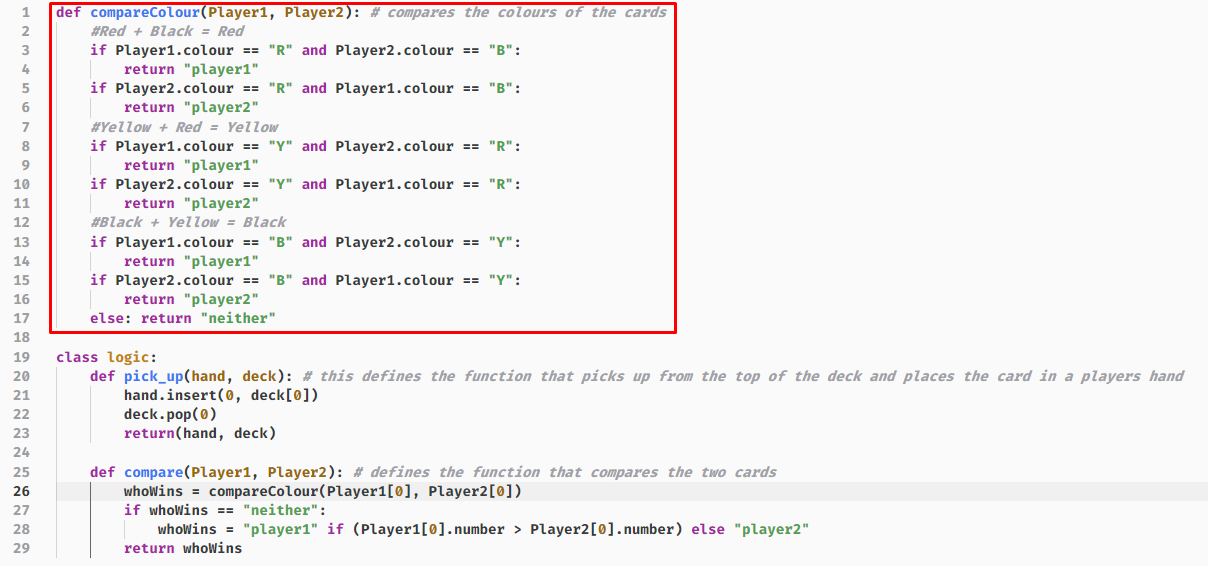
game.py – is where the code for the game resides

betterinput.py – is where the input functions I have created go

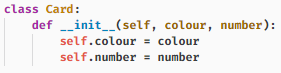
main.py – imports game.py and input capture and at the moment includes the main menu system

**card.py**

Contains the logic for comparing and managing the cards, I laid this out by creating a class called logic which contains the functions used by other python files (is imported in ‘game.py’) and then defining the function compareColour separately. This means that it isn’t imported unnecessarily into other python files.



This file also contains the card class



**game.py**

Contains a function which starts and runs the game. It has random imported and has the Card and logic classes imported.



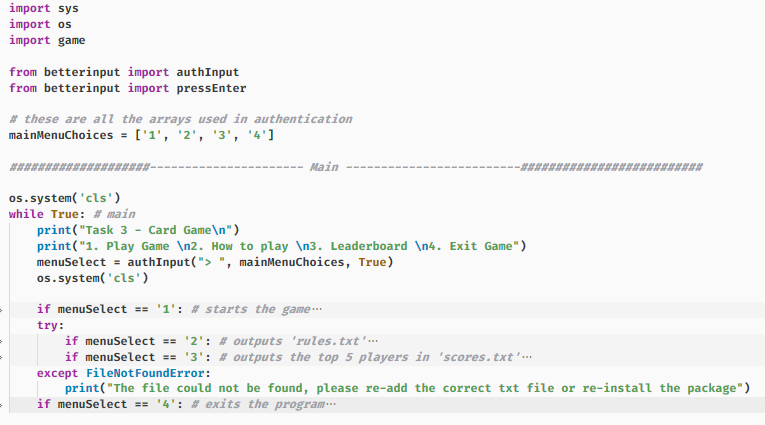
**betterinput.py**

This only contains two functions but both are used in multiple places in the project. authInput has already been described and pressEnter just allows for a better user experience by clearing the screen when the user presses enter when prompted.

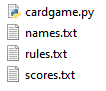


**\_\_main\_\_.py**

This contains the main menu and is where the program is ran from!

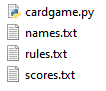


I also found a really cool feature which makes distributing all these files a bit easier! When you rename your main python file to \_\_main\_\_.py and place all of the needed python files into a folder, you can run the ‘directory’ from the command line  like this and python will run the \_\_main\_\_.py file instead of spitting up a bunch of errors.

This also works if you zip the file. So knowing this, I tried zipping the python files into a .zip file then renaming the resulting zip file to ‘cardgame.py’.

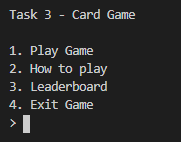
This means that windows would assume that the .zip file was a python file and would run it as if it was a python file. It would also appear to most users to be just an ordinary python file (which isn’t editable by IDLE or any python IDE) instead of a zip file meaning the source code is more difficult to find which makes it more secure.

Obviously the user could just unzip the python file and it isn’t anywhere near as secure (or portable) as it would be if I’d used something like pyInstaller or cython, which don’t need the python interpreter installed on the system to run the outputted files. But I like the idea of all the files being kept in one ‘python file’

**EDIT:** Turns out, this works, but the file handling won’t work. Since windows uses the python.exe when you run python, the current working directory will be in the wrong place entirely so it cannot access any of the necessary files! This functionality is still possible, but the only way to run it is if you run it straight from command line, open the file straight from the windows file explorer or use a batch file.

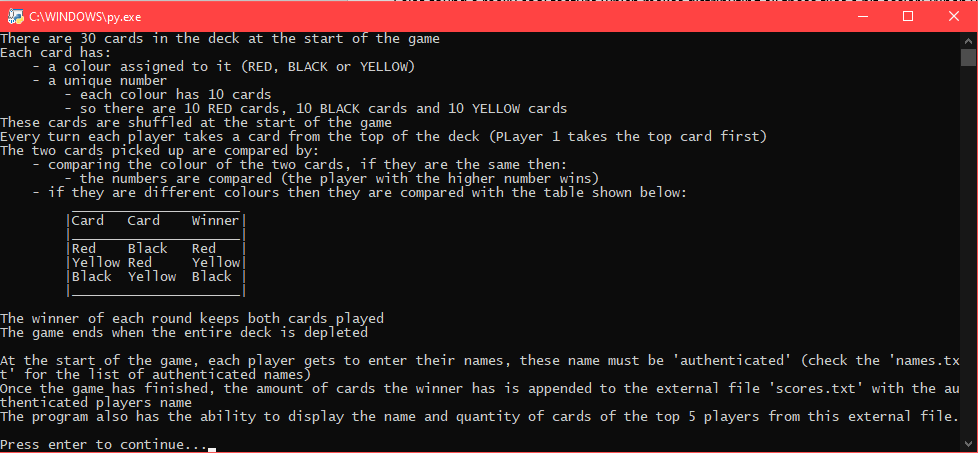
**Evaluation**

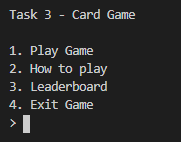
So, if I open the cardgame.py file here:

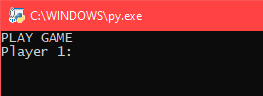
The menu system opens up. By typing the number of the area of the program that the user wants to open, they can choose to:

* Play the game
* View the rules
* View the current Leaderboard
* Exit the game

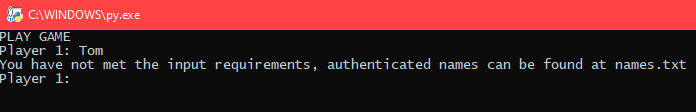
If the user wants to read the rules, they enter 2.



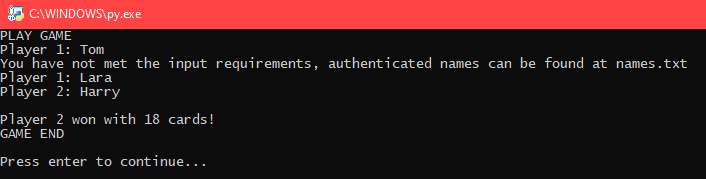
This reads the text file rules.txt and outputs it to the user in the terminal. Then when the user has finished reading the rules they can press enter to return to the main menu.

If the user wants to play the game, they should enter ‘1’

This prompts the user to enter the name of player 1. If the user enters a name that isn’t authenticated, then this error message pops up and the user is asked to enter the name again:

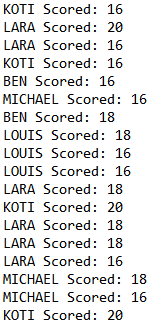


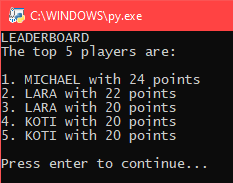
When they type in the correct name they are then prompted to enter the name of player 2. The same process occurs if player 2’s name is not authenticated. In this example we will enter a correct name to save time.

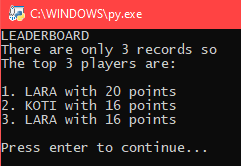


As you can see, I have entered player 1 and 2’s names. Then the program completes all of the games necessary calculations to determine and output the winner and their card count.

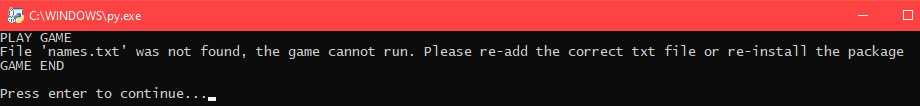
To make it easier for the user to determine when the game starts and ends, the program outputs when the game function has begun and ended in ‘PLAY GAME’ and ‘GAME END’.

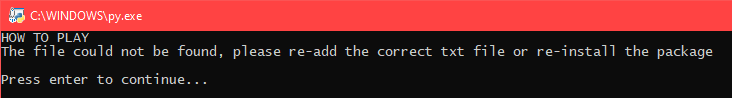
Now the game has finished, the user is prompted to press enter to continue. This returns the user to the main menu where they can choose option 3 to view the leaderboard.

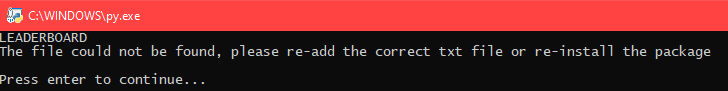
This searches through the file ‘scores.txt’ (right) and sanitizes the data so the program can calculate the top 5 scorers in the text file. It then outputs each of the players’ names along with their scores (left).

If the total number of records of winners in ‘scores.txt’ is lower than 5 then the program can detect this and accommodate for the potential error, outputting the image on the left.

As you can see, the program recognizes that there are only 3 records so states this and outputs the top 3 players instead.

Once the user has finished looking at the leaderboard, they can press enter to return to the main menu. If the program cannot find any of the ‘\*.txt’ files in its current directory then these messages are output: 





Alright, this is the end of my programming project! The project runs according to the specification and I have added tons of functionality to the project as well, I have learnt a lot about how to structure my programs in the future and have built a program that I am proud of!